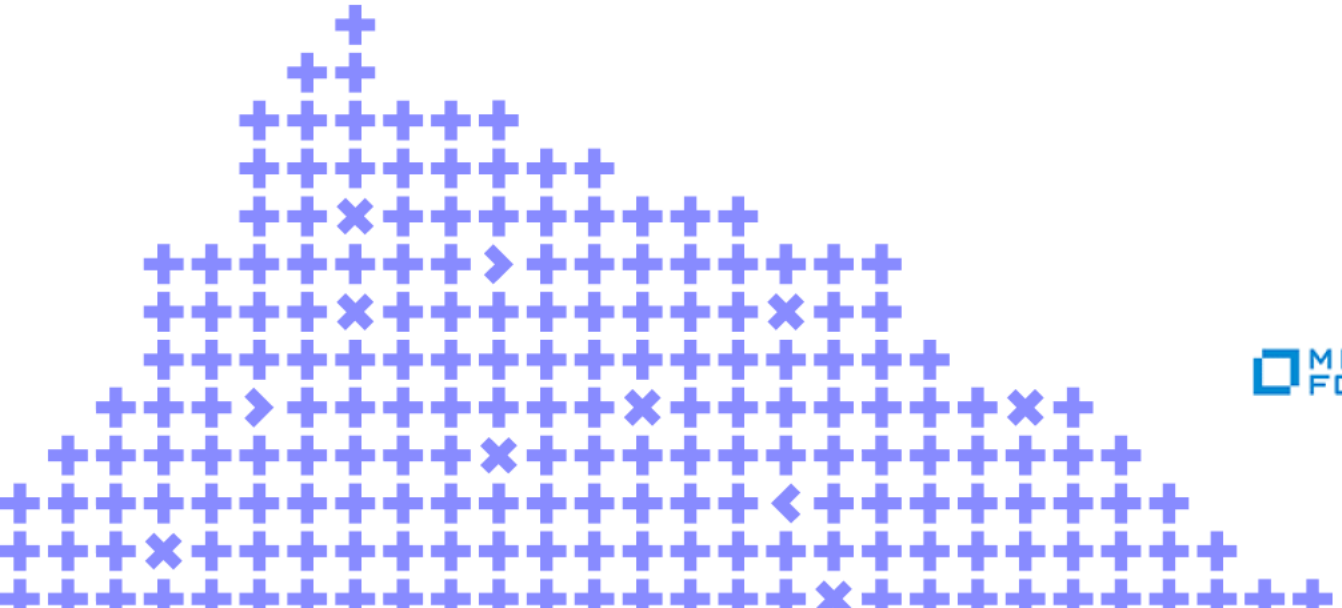
Designing a more efficient OLAP database data flow and architecture

Chris Bohn

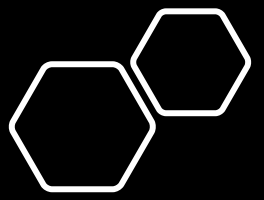


MICRO FOCUS | VERTICA

High Load ++ Armenia

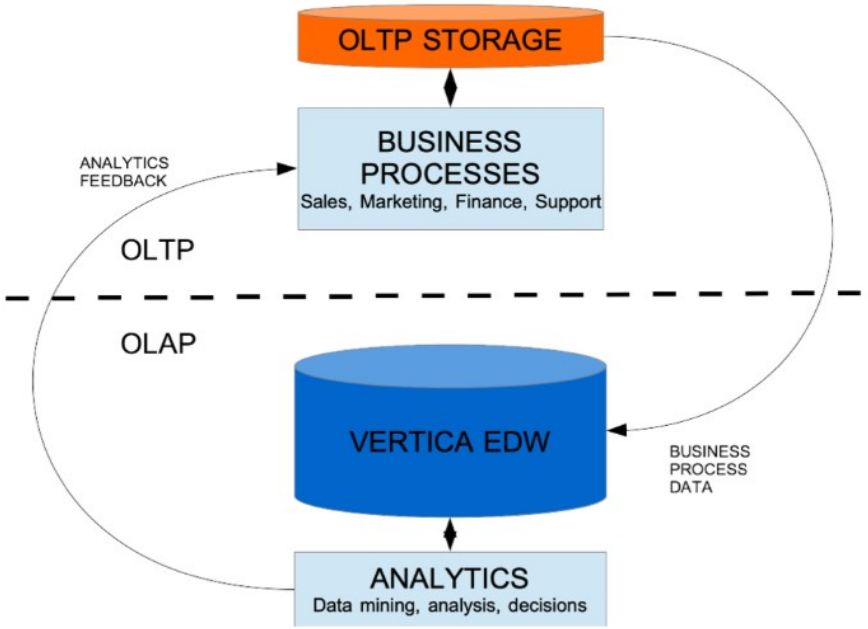
Co-organizer

Yandex



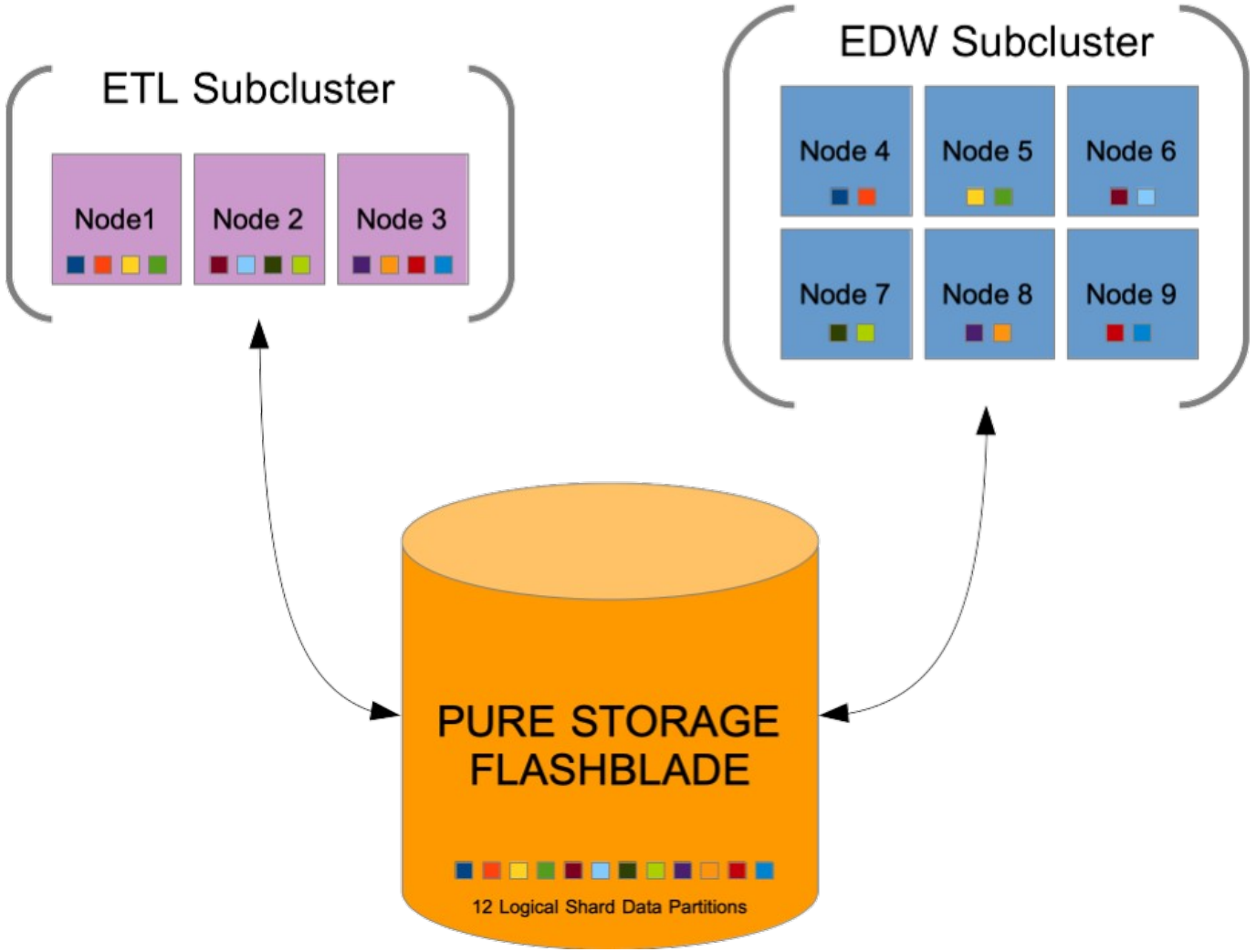
Holistic OLTP/OLAP

- CB Bohn
- Vertica
- cb@vertica.com

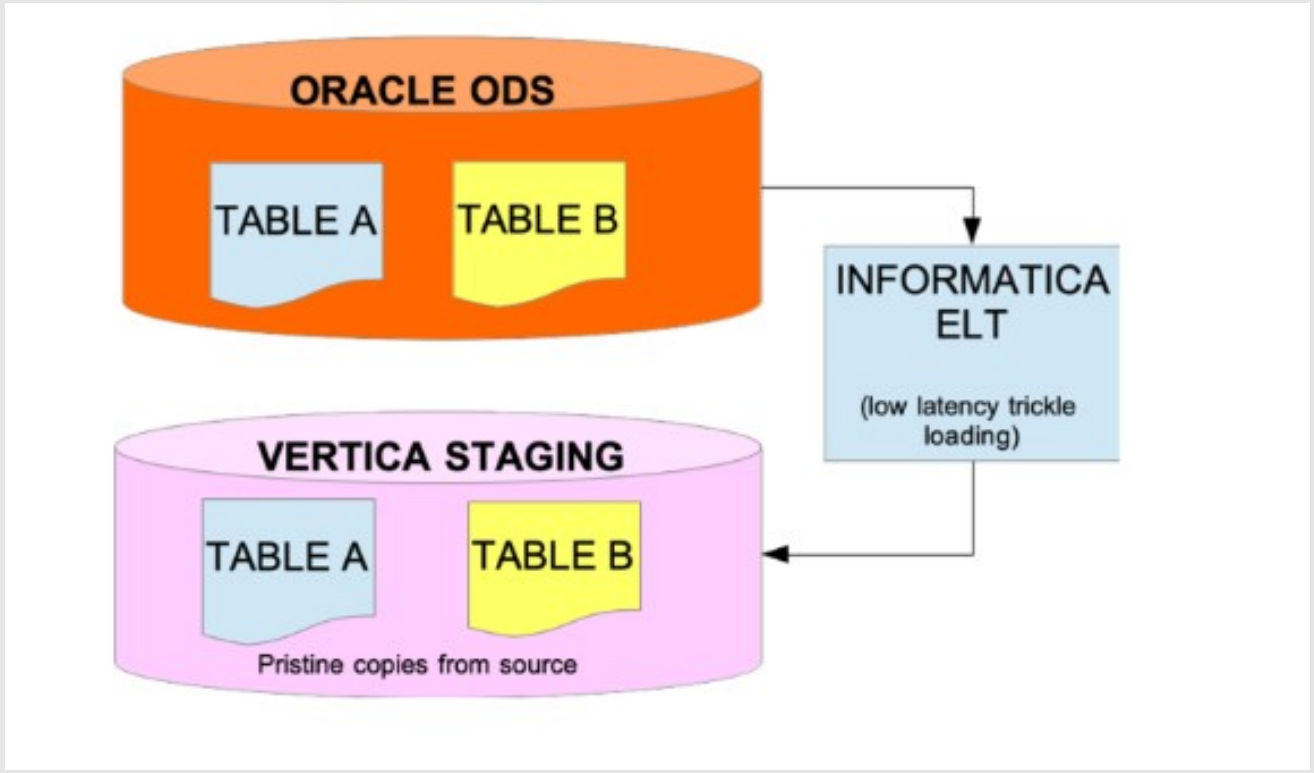


MICRO FOCUS | VERTICA

MicroFocus EDW Topology (Vertica)

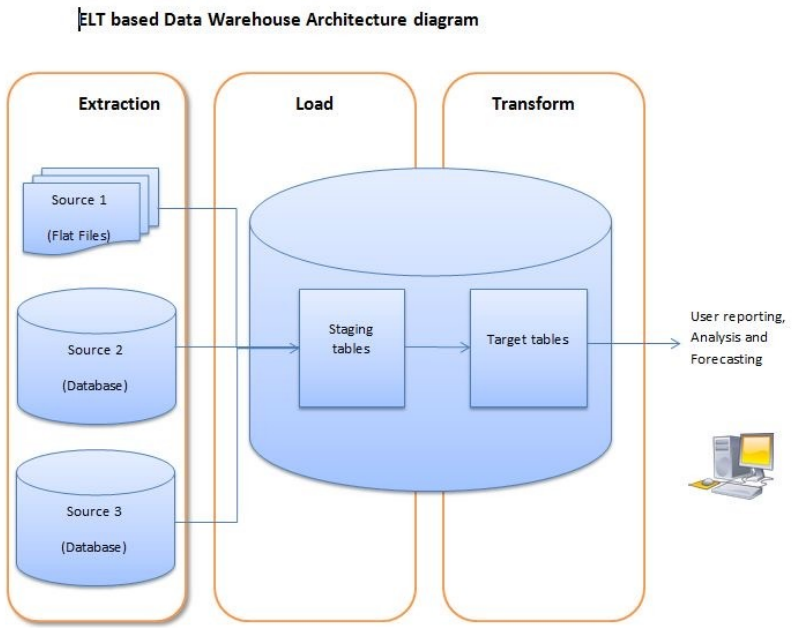


MicroFocus ELT

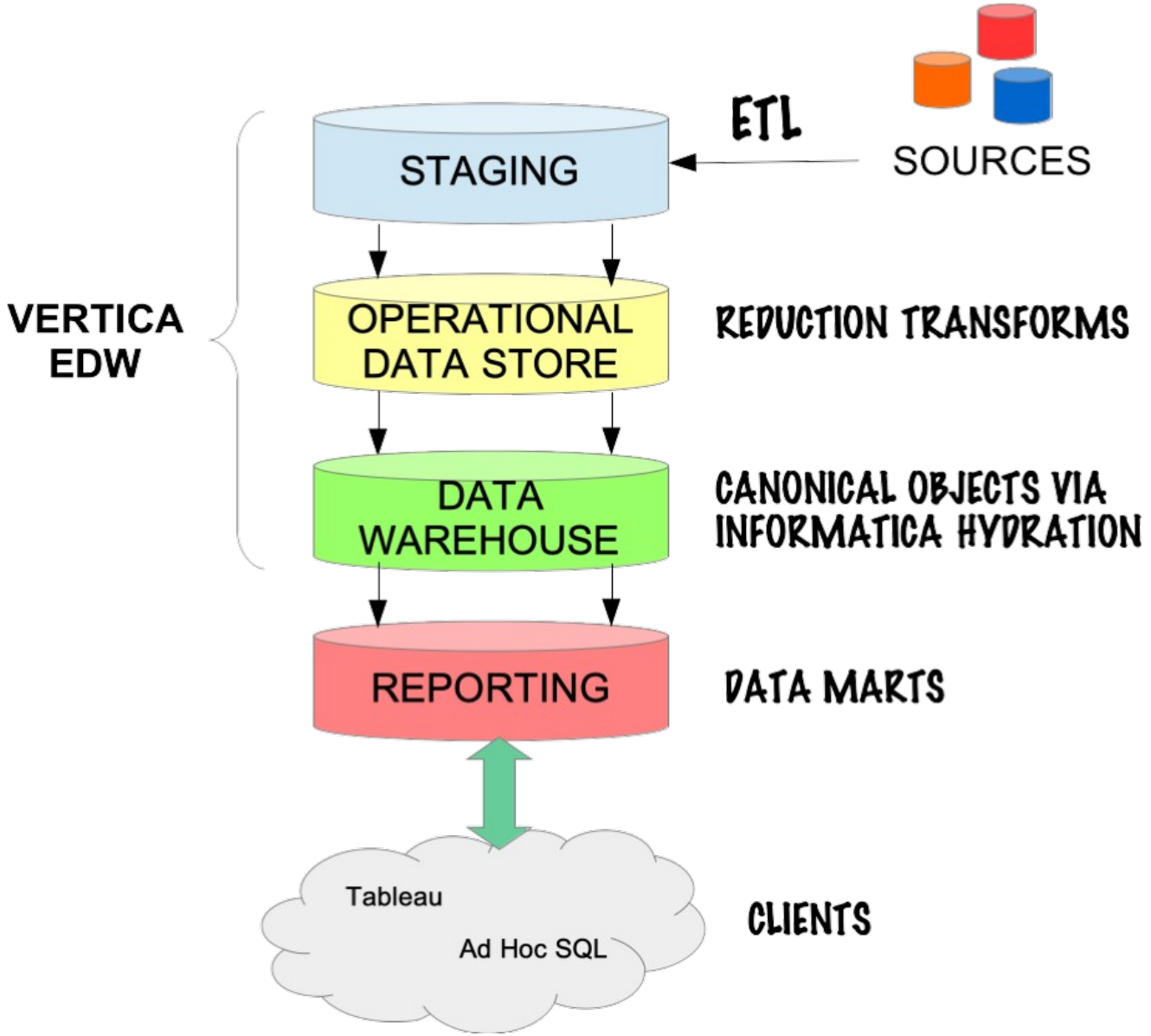


- **Append-only data loading**

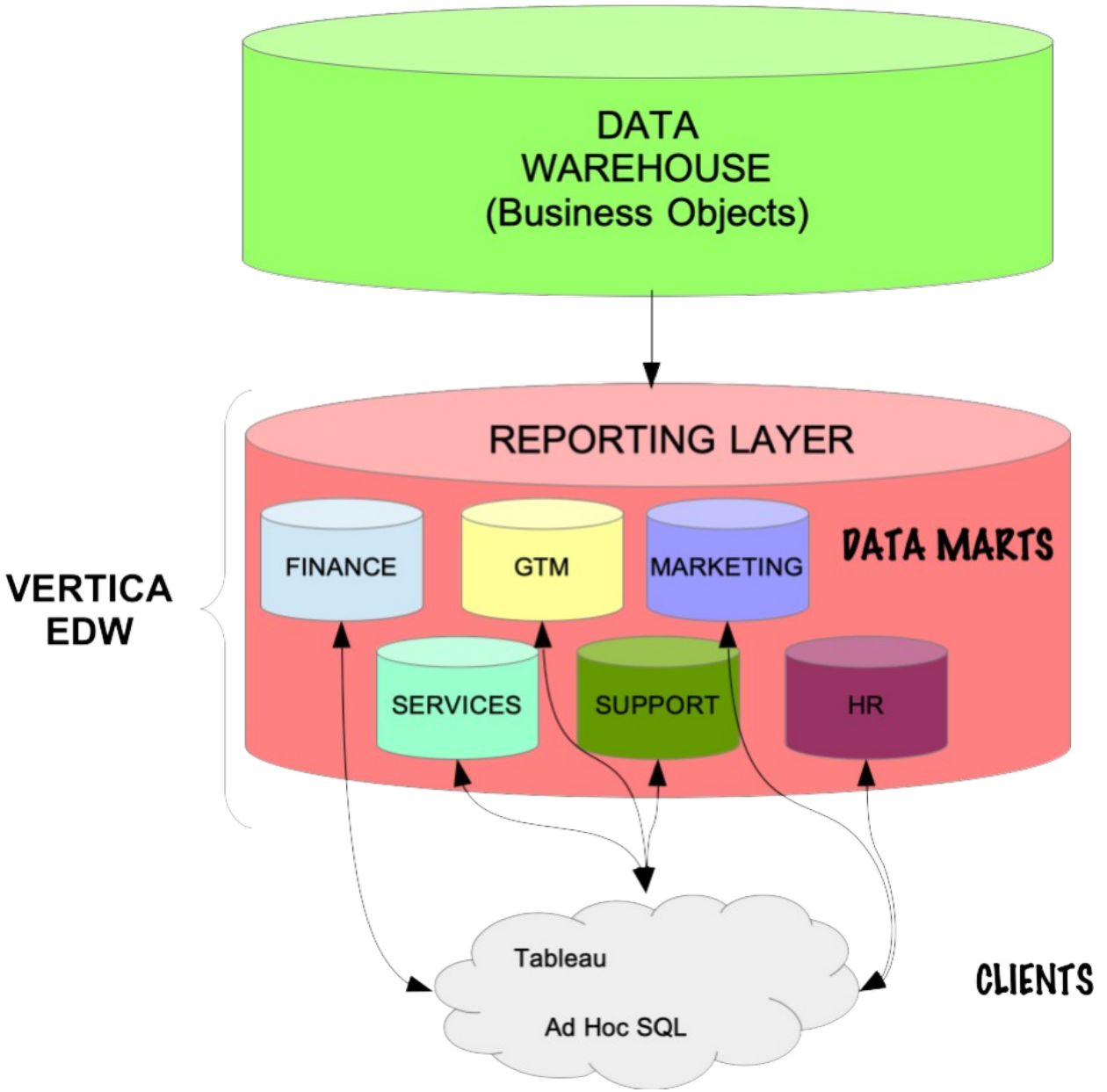
Classic ELT for EDW Architecture



MicroFocus EDW Data Flow

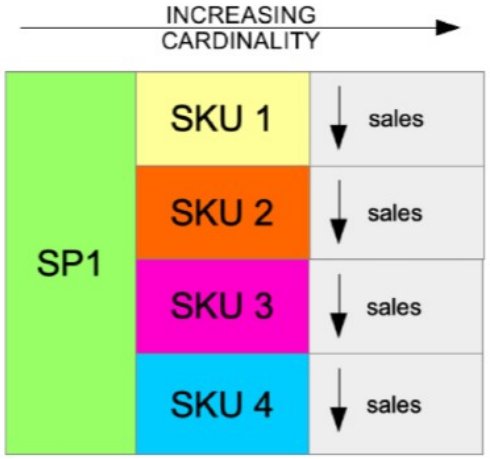


MicroFocus EDW Reporting Layer

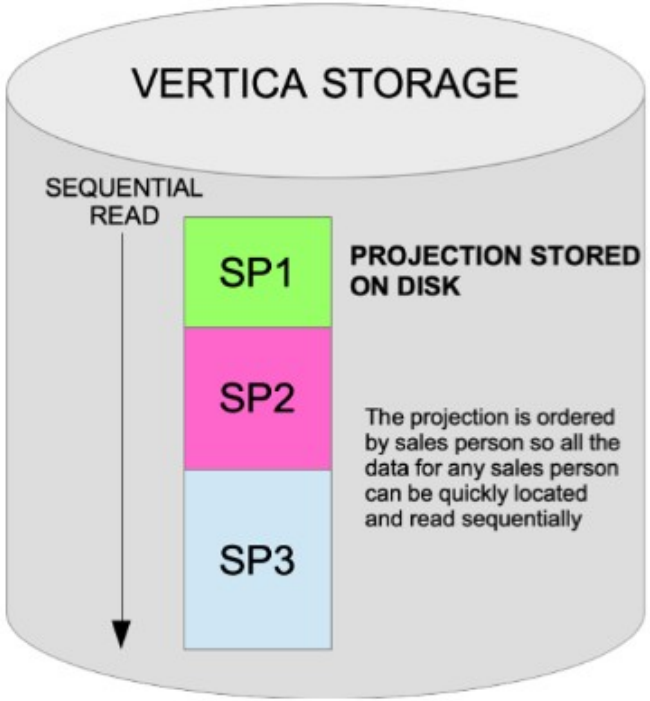


Columnar Store (C-Store)

- Developed at MIT

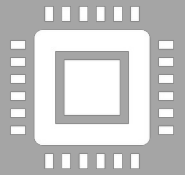


The projection creates quick *pathways* to access sales data for each SKU for this sales person. This enables Vertica to speedily aggregate totals, because the relevant data is clustered together and can be read sequentially.



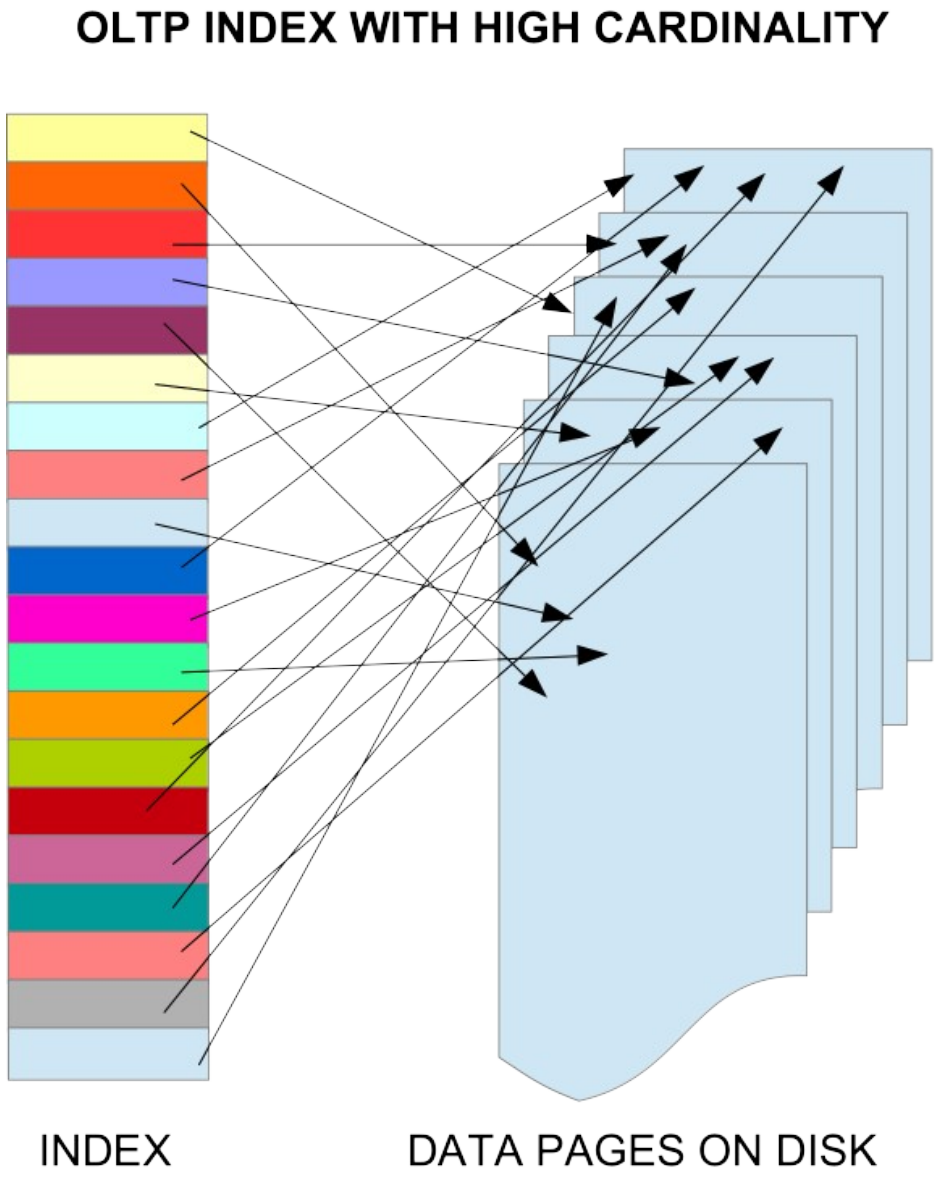
How OLTP and OLAP Databases Are Optimized

OLTP: uses indexes for fast lookup – best on high cardinality columns

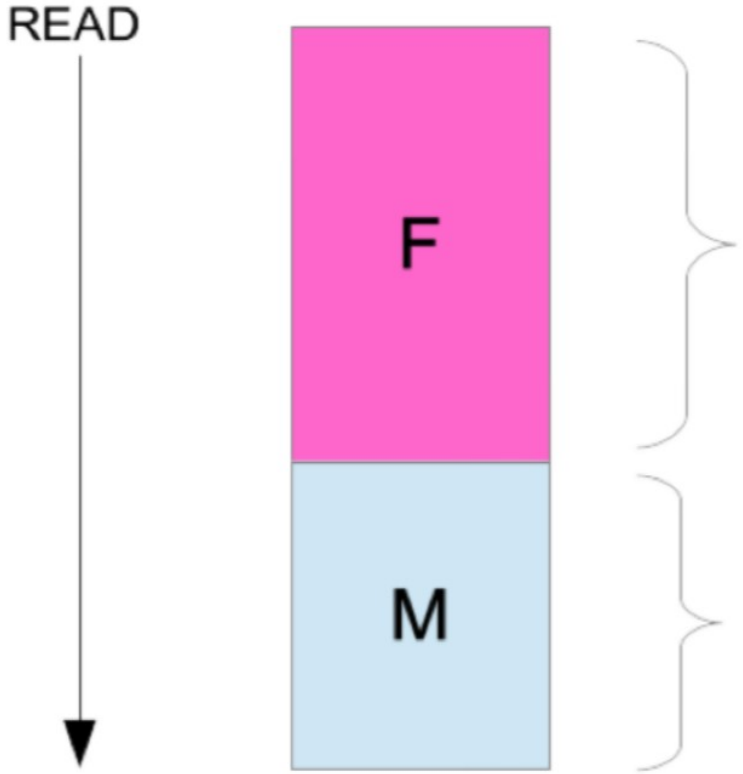


OLAP: generally no indexes; instead, aggregation speed achieved via ordered columnar storage and data encoding

OLTP Indexes on High Cardinality Columns



OLAP (Vertica) Run Length Encoding



Due to run-Length encoding (default), Vertica knows that there are 600K entries which have the FEMALE gender attribute. So a COUNT(*) with a correlation on gender = 'F' is almost instantaneous, no sequential scan required. Since we are only sorting and ordering by gender, the other field attributes are sequentially encountered randomly.

What is said above for the F gender applies here also to the M gender. Due to RLE encoding, Vertica already knows that there are 400K records with gender = 'M'. All further field attributes are sequentially encountered randomly.

OLTP vs OLAP = Fast Lookup vs Fast Aggregation

Indexes are great for high cardinality data (primary keys are highest possible cardinality); indexes on low cardinality columns are often ignored resulting in full sequential scans.

Columnar storage is best when ordered by lowest cardinality to highest; allows for fast aggregation. It is not efficient for specific record lookup.

OLTP and OLAP Query Planners

Query Planners (QP) decide the best and fastest way to process a query

QPs examine query predicates for clues on optimization choices

For OLTP databases, QP picks the best index(es) to use; QP can and will ignore indexes that provide no benefit such as indexes on low-cardinality data, since sequential scan may be just as fast.

For OLAP databases, QP picks best columnar store to use; QP will default to base columnar store if a better one for the query does not exist.

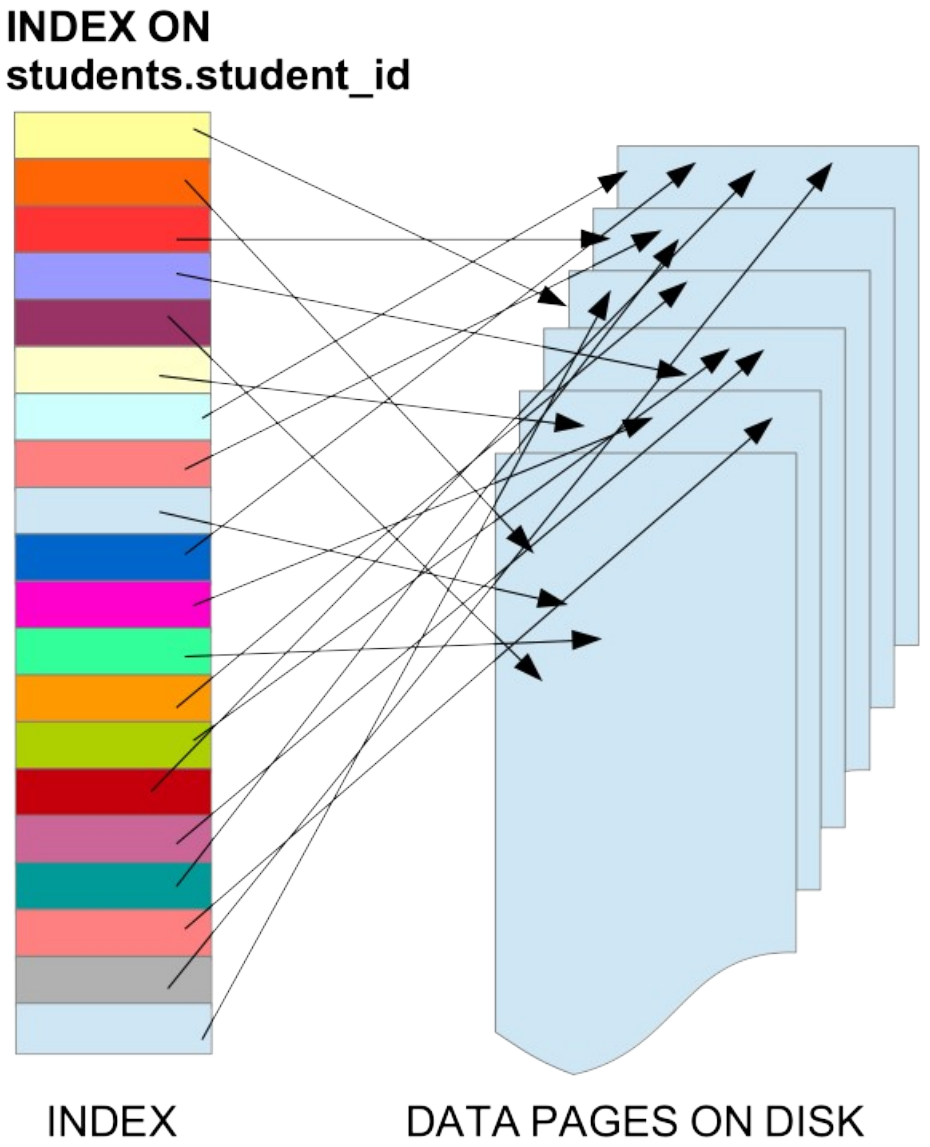
DBA CHALLENGE: Creating indexes and columnar stores optimized for fast lookup or aggregate queries.

Sample Query: OLTP vs OLAP

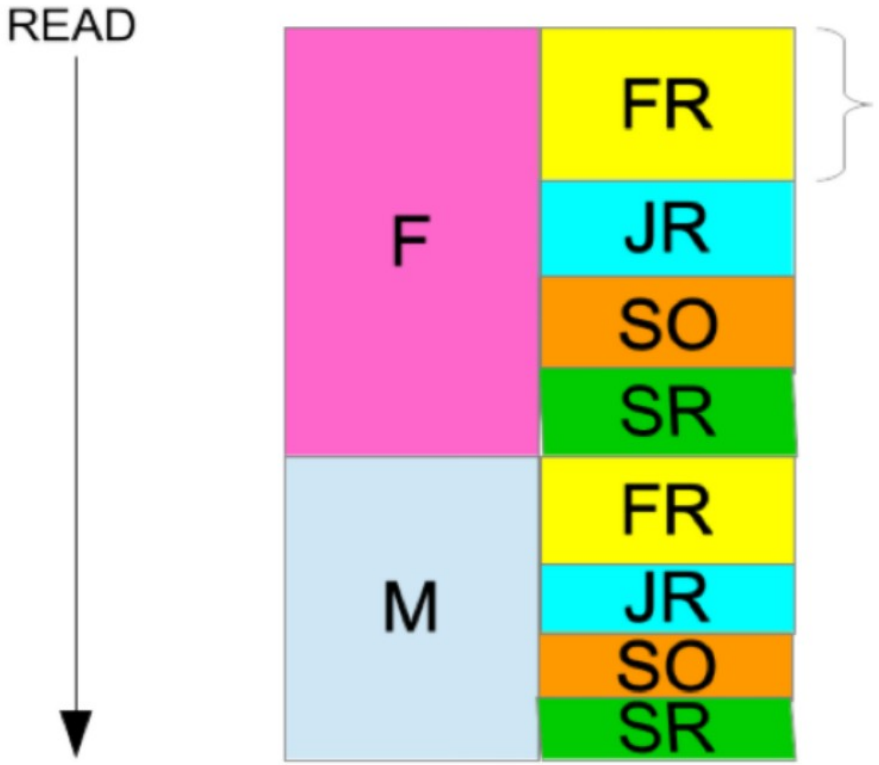
```
SELECT COUNT(*) FROM students
WHERE gender = ‘F’ AND CLASS =
‘JR’;
```

OLTP Query Processing

- **Index does not contain any gender or class information. Thus, all data pages must be read and filtered.**



OLAP FTW: Adjacent RLE columns



Now the data is sorted on class as well as gender. The data beyond gender is not random as it was before – it is now sequentially ordered by class as well. So as the data is read sequentially, Female Freshman are first clustered together, followed by Female Juniors, etc. Due to RLE, the total count of Freshmen who are female is already known. The same applies to all other classes, and to the “M” gender and its classes, too. However, any further field attributes are random (such as student_id)

Problems with OLTP

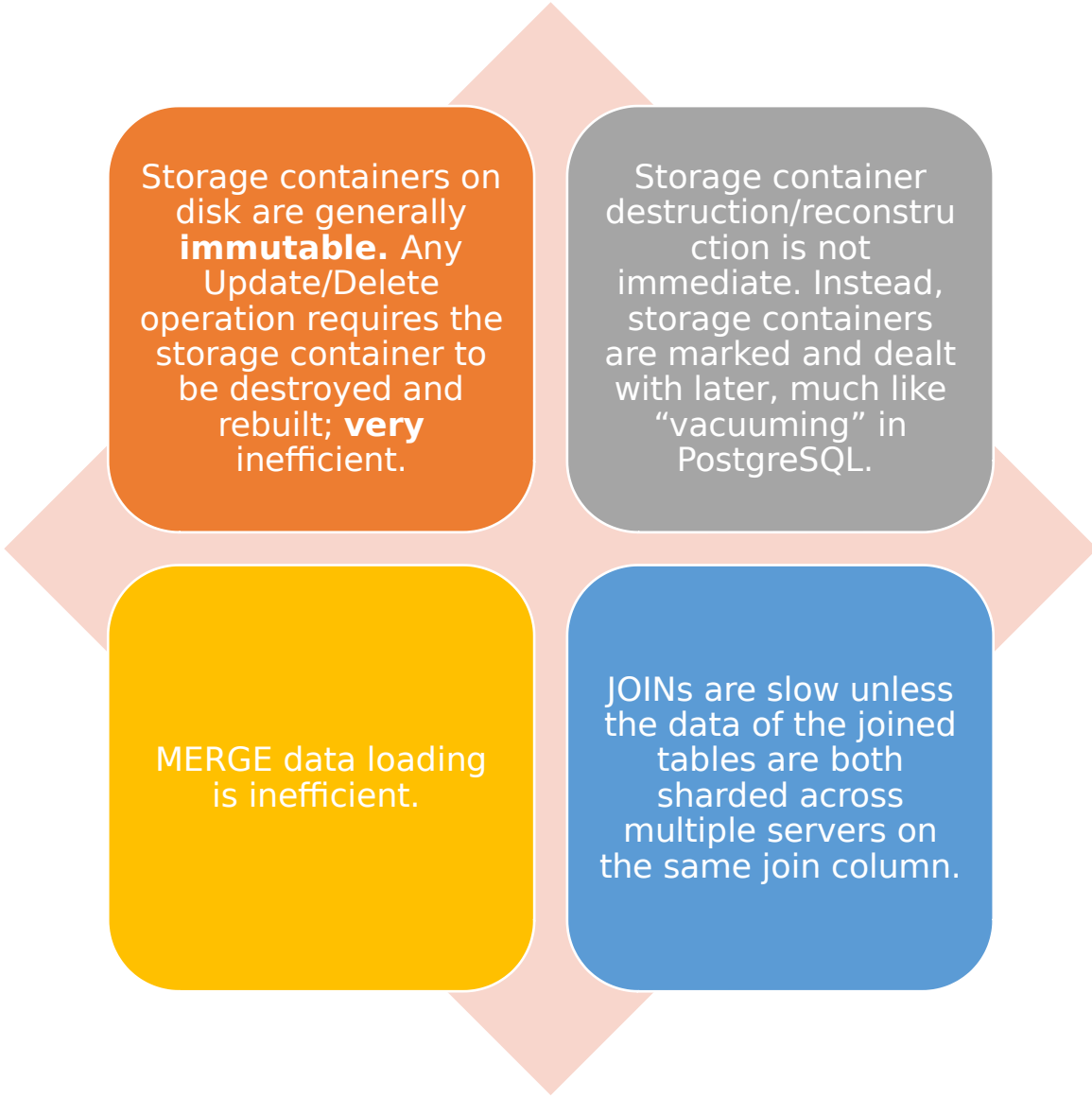
Oriented and optimized for transactions involving specific records identified by primary keys and other indexes on high-cardinality columns.

Not an efficient design for aggregation and analytics because the database query planner will ignore indexes on low cardinality columns, resulting in sequential scans. This is why OLTP databases do not perform well in BI roles.

Problems with OLAP

- Designed for fast aggregation queries
- No indexes, instead columnar data ordering and encryption is used
- Run-length encoding (RLE) is common for low cardinality columns. Example column is “gender” which is mostly 50/50 distribution M/F.
- SELECT gender, count(*) FROM users GROUP BY 1; is a slow query on OLTP, because gender column cannot be effectively indexed (low cardinality). But very fast in OLAP due to ordered columnar storage and RLE.
- OLAP stores data in immutable “storage containers.” Any update or delete means that the storage container has to be destroyed and recreated. Very inefficient!

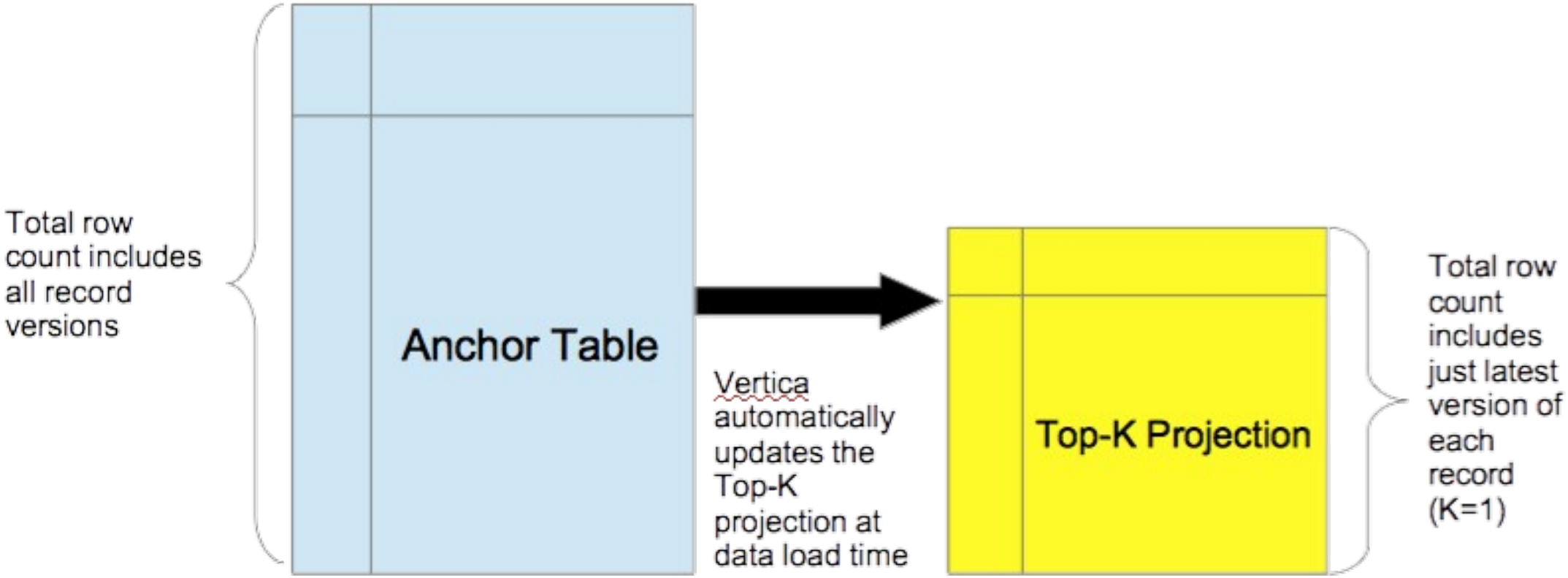
OLAP Performance Killers



OLAP Best Optimized by Append-Only ELT

- Databases originally designed with update/delete capability to reduce storage costs which were very expensive in 1970s
- Due to Moore's Law, storage costs have decreased dramatically
- Analytic tools like Vertica have increased value of data
- Change history now a critical aspect of data analytics
- Use SQL techniques to generate "latest snapshot" of table data
- Preserves change history
- Reduces storage container "churn"
- Best if OLTP does "soft deletes" so that it is also essentially "append only"

Transforming Staging Table to Latest Snapshot



Coding Top-K Projections

```
CREATE PROJECTION proj-name [(proj-column-spec)]
      AS SELECT select-expression FROM table
      LIMIT num-rows OVER (PARTITION BY expression ORDER BY column-expr);
```

For the latest record, K=1. Here is how we would create a Top-K projection for listings (Note
have simplified the column list for clarity):

```
CREATE PROJECTION listings_latest
      AS SELECT listing_id, update_date, title FROM table
      LIMIT 1 OVER (PARTITION BY listing_id ORDER BY update_date
DESC);
```

So how do we craft a query to select from the Top-K projection? Like this:

```
SELECT listing_id, update_date, title
FROM listings
LIMIT 1 OVER (PARTITION BY listing_id ORDER BY update_date DESC);
```

Wrapping Top-K Projections with Views

- Simplify queries by using views to wrap Top-K projections; removes the Top-K code

```
CREATE VIEW v_listings_latest AS
    SELECT listing_id, update_date, title
    FROM listings
    LIMIT 1 OVER (PARTITION BY listing_id ORDER BY update_date DESC);
```

Now to get the latest state of all listings, we just need query the view, without the need to work in that LIMIT 1 OVER()... predicate:

```
SELECT * FROM v_listings_latest;
```

Of course, individual records can also be pulled from the view:

```
SELECT * FROM v_listings_latest WHERE listing_id = 12345;
```

Change History – FOR FREE!

"Change History" means keeping the history of all changes over the lifetime of all records

"Change History" has become a very important analytics tool

It is used to correlate changes over time to performance over time

Because all data via ELT into MicroFocus EDW IS APPEND-only, by definition, the initial, pristine staging table contains **all** versions of records, and as such is the full Change History. We use the view wrapping the Top-K projection to get latest snapshot.

Referential Integrity

Referential integrity is important for maintaining clean data. But where is the best place to do that?

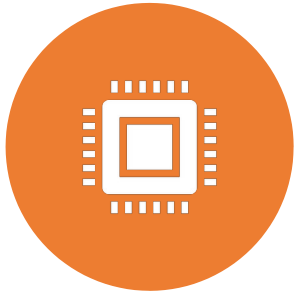
OLTP databases are the best place to specify and enforce referential integrity, due to efficient indexing of columns.

OLAP databases essentially **inherit** the referential integrity enforced upstream in OLTP systems. **No need to repeat!**

The inheritance creates “eventually consistent” referential integrity, akin to “eventually consistent” replication.

MicroFocus EDW (Vertica) sources data via ELT from Oracle, SQLServer, SalesForce and clickstream and other logs. No primary or foreign keys are used, we rely on and trust upstream referential integrity enforcement. Works great!

Benefits of Eschewing Referential Integrity Constraints in OLAP Databases



Why redo in OLAP what has already been done upstream in OLTP?

Makes ELT easier because the order of data arrival is not important: “Eventually Consistent Referential Integrity”.

High velocity fact tables can be loaded more frequently than slowly changing dimension tables, without breaking things. Again, due to “Eventually Consistent Referential Integrity”.

Only consideration is that we must query after all lag time is taken into consideration, such as end-of-quarter summations.

Summary of Best Practices for OLTP and OLAP

Design	If possible, design OLTP data to feature soft deletes instead of hard
Use	Use append-only ELT data loading into OLAP database
Don’t Use	No need to use MERGE commands
Use	Use Top-K projections (or equivalent) to generate materialized views of snapshots of latest states of tables in OLAP database
Design Columnar Stores with low-cardinality	Design OLAP columnar stores (projections) with left-most columns having lowest cardinality (inverse of indexes in OLTP databases)
Use	Never use SQL “DELETE” command – causes too many delete vectors. Instead, use TRUNCATE to empty a table, or drop partitions
Make	Change History is a no-cost benefit; make analysts aware of this!

Questions ?